Uthreads Library

Jace Halvorson Ryan Johnsen

Assumptions:

- We assumed that disableInterrupts() and enableInterrupts() would execute atomically

Pi:
Used to test basic library functions like uthread_init, uthread_create, and uthread_join. The program itself calculates π using random numbers.

- uthread_init()
- uthread_create()
- uthread_join()

Expected output: It should print to the console when threads are being created and when they are being joined. It will also print the computed π.

Test Case 4:
Used to test other supporting library functions. It uses 20 threads to calculate how many prime numbers are less than 10,000. A third of the threads exit after 500 prime numbers, and after joining all threads, each thread's result is added together.

- Uthread_suspend()
- uthread_resume()
- uthread_exit()
- uthread_get_quantums()
- uthread_get_total_quantums()

Test Case 5:
Used to mainly test uthread_yield, but uses other basic library functions. It uses 50 threads to calculate how many prime numbers are less than 1000. Threads with thread ids divisible by 8 are yielded when they compute a prime number. The results are added together in the end similar to test case 4.

- uthread_yield()

Test Case 6:
Tests maximum capacity of threads by counting in a triple for loop a set number of times dependent on the thread ID. Attempts to create a 100th thread and expects an error message. Joins threads and combines their return values.

How did your library pass input/output parameters to a thread entry function? What must makecontext do “under the hood” to invoke the new thread execution?

Input parameters are passed as raw bytes by casting the data to a void*. The output parameters are retrieved by the calling thread using a void** that is allocated by the calling thread to get the outputs. Makecontext creates a new stack and return address in the operating system.